

06 | XSS：当你“被发送”了一条微博时，到底发生了什么？

2019-12-20 何为舟

安全攻防技能30讲

[进入课程 >](#)



讲述：何为舟

时长 20:31 大小 18.80M



你好，我是何为舟。

在前面的课程中，我们重点讲解了安全的一些基础知识，更多地是从宏观的层面上来谈论安全。但**安全不是一个靠宏观指导就能够落地的东西**。因此，接下来我会结合真实案例中的各种安全问题，来介绍具体的安全防护手段和工具。今天，我们就先从最基础的 Web 安全开始。

在 Web 安全这个模块中，我们所谈论的 Web，是指所有基于 HTTP 或者其他超文本传输协议（RPC 等）开发的应用，包括：网页、App、API 接口等等。这类应用的共同点是：通过 HTTP 等文本协议，在客户端和服务端之间进行数据交换。客户端需要将服务端传出

的数据展示渲染出来，服务端需要将客户端传入的数据进行对应的处理。而 Web 安全所涉及的正是这些应用中存在的各类安全问题。

背景介绍完了，下面我们进入今天的正题。

基于前面安全基础知识的学习，你现在通过了面试官的考核，成功进入了这家公司。某一天，公司的网页应用中发生了一件事。

有很多用户发送了同样类型的内容，而且这些内容都是一个带有诱惑性的问题和一个可以点击的链接。这些用户全部反馈说，这不是他们自己发的。前端开发表示，用户内容都是后端产生的，他不负责。后端开发表示，这些内容都是用户自己提交上来的，他也不负责。正当大家议论纷纷的时候，你作为学习过安全专栏的人，敏锐地发现了问题的原因：这是黑客发起了 XSS 攻击。

这个事情的原型，其实是 2011 年微博真实出现的一次安全事件。整个事件的核心问题，其实出在这个可以点击的链接上。在这个事件中，黑客并不需要入侵到微博服务器中，只要用户点击了这个链接，就会“被发送”这样的博文。

这就是著名的 XSS 攻击所能够实现的效果。那么，XSS 攻击究竟是怎么产生的呢？我们究竟该如何防护呢？今天我就带你来了解这个网页中最经典的 XSS 攻击。

XSS 攻击是如何产生的？

首先，我们来看，XSS 攻击是如何产生的。作为最普遍的网页语言，HTML 非常灵活，你可以在任意时候对 HTML 进行修改。但是，这种灵活性也给了黑客可趁之机：通过给定异常的输入，黑客可以在你的浏览器中，插入一段恶意的 JavaScript 脚本，从而窃取你的隐私信息或者仿冒你进行操作。这就是 **XSS 攻击**（Cross-Site Scripting，跨站脚本攻击）的原理。

你现在应该对 XSS 有了一个大致的了解，除此之外，你还需要了解三种 XSS 攻击，它们分别是：反射型 XSS、基于 DOM 的 XSS 以及持久型 XSS。下面我们——来看。

1. 反射型 XSS

假设现在有一个搜索网页，当你输入任意一个关键词，并点击“搜索”按钮之后，这个网页就会给你展示“你搜索的结果内容是：XXX”。

搜索

你搜索的结果内容是：XSS

我们以 PHP 为例，这个网页的服务端实现逻辑如下所示：

 复制代码

```
1 <!DOCTYPE html>
2 <html>
3   <body>
4     <form role="search" action="" method="GET">
5       <input type="text" name="search" placeholder=" 请输入要搜索的内容 ">
6       <button type="submit"> 搜索 </button>
7     </form>
8     <?php
9       if (isset($_GET['search']) && !empty($_GET['search'])) {
10         $search = $_GET['search'];
11         echo "<h3> 你搜索的结果内容是： " . $search . "</h3>";
12       }
13     ?>
14   </body>
15 </html>
16
```

我们可以看到，这段代码的逻辑是将搜索框输入的内容，拼接成字符串，然后填充到最终的 HTML 中。而且这个过程中没有任何的过滤措施，如果黑客想要对这个过程发起攻击，他会输入下面这行代码：

 复制代码

```
1 </h3><script>alert('xss');</script><h3>
```

黑客输入这段字符后，网页会弹出一个告警框（我自己测试的时候，发现部分浏览器，如 Safari 不会弹出告警框，这是因为浏览器自身提供了一定的 XSS 保护功能）。



通过查看网页的源码，可以发现，这中间多了一段 JavaScript 的脚本：

```
<html> == $0
<head></head>
▼ <body>
  ▶ <form role="search" action method="GET">...</form>
    <h3>你搜索的结果内容是: </h3>
    <script>alert('xss');</script>
    <h3></h3>
  </body>
</html>
```

这就是我们所说的反射型 XSS 攻击的过程。其实它攻击的原理很简单。我们可以总结一下，即通过开头的</h3>和结尾的<h3>，将原本的<h3>标签进行闭合，然后中间通过<script>标签插入 JavaScript 代码并执行，就完成了整个反射型 XSS 的流程。

你可以注意一下浏览器的地址：[http://localhost/index.php?search=<%2Fh3><script>alert\('xss'\)%3B<%2Fscript><h3>](http://localhost/index.php?search=<%2Fh3><script>alert('xss')%3B<%2Fscript><h3>)。实际上，任何人只要点击了这个链接，就会执行一段黑客定义的 JavaScript 脚本。所以，我们经常说，不要点击任何未知的链接。

反射型 XSS 的总体流程我总结了一下，你可以看下面这张图。黑客诱导你点击了某个链接，这个链接提供的服务，可能就是上述的搜索功能。网页在解析到链接的参数后，执行正常的搜索逻辑，但是因为漏洞，网页中被填入了黑客定义的脚本。使得用户的浏览器，最终执行的是黑客的脚本。



2. 基于 DOM 的 XSS

在上面的例子中我们可以看到，反射型 XSS 产生在前后端一体的网页应用中，服务端逻辑会改变最终的网页代码。但是，目前更流行的其实是前后端分离，这样网页的代码不会受服务端影响。那么，这样是不是就安全了呢？

显然不是的。尽管服务端无法改变网页代码，但网页本身的 JavaScript 仍然可以改变。而黑客只要利用了这一点，同样能够在网页中插入自己的脚本。这也就是所谓的基于 DOM 的 XSS 漏洞。

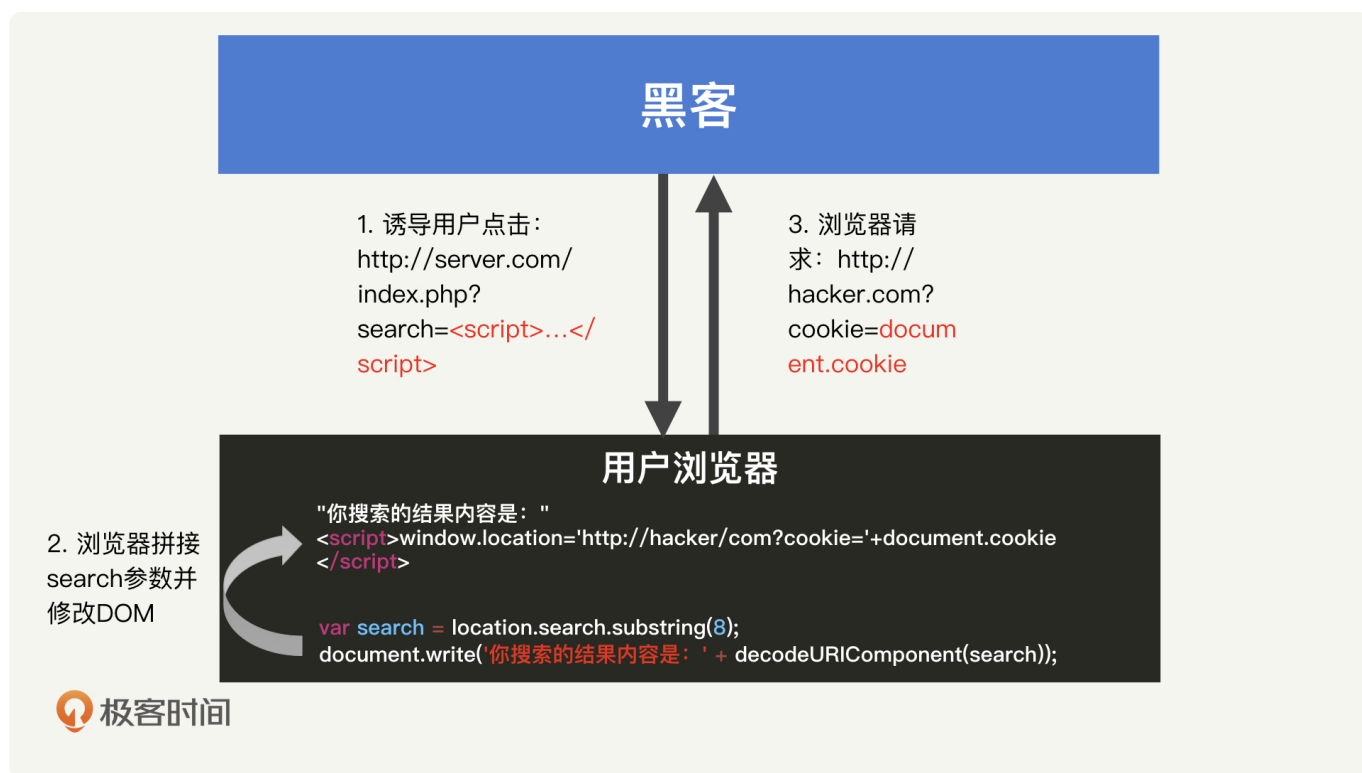
对于上述搜索功能，通过前后端分离，它的源码就变成了下面这样：

复制代码

```
1 <!DOCTYPE html>
2 <html>
3   <body>
4     <form role="search" action="" method="GET">
5       <input type="text" name="search" placeholder=" 请输入要搜索的内容 ">
6       <button type="submit"> 搜索 </button>
7     </form>
8     <script>
9       var search = location.search.substring(8);
10      document.write('你搜索的结果内容是: ' + decodeURIComponent(search));
11    </script>
12  </body>
```

这段代码能够实现和之前的 PHP 代码相同的逻辑：当你在搜索框点击搜索关键词之后，网页会展示你输入的关键词。只不过，HTML 是通过 JavaScript 脚本修改 [DOM](#) 来实现这个功能的。

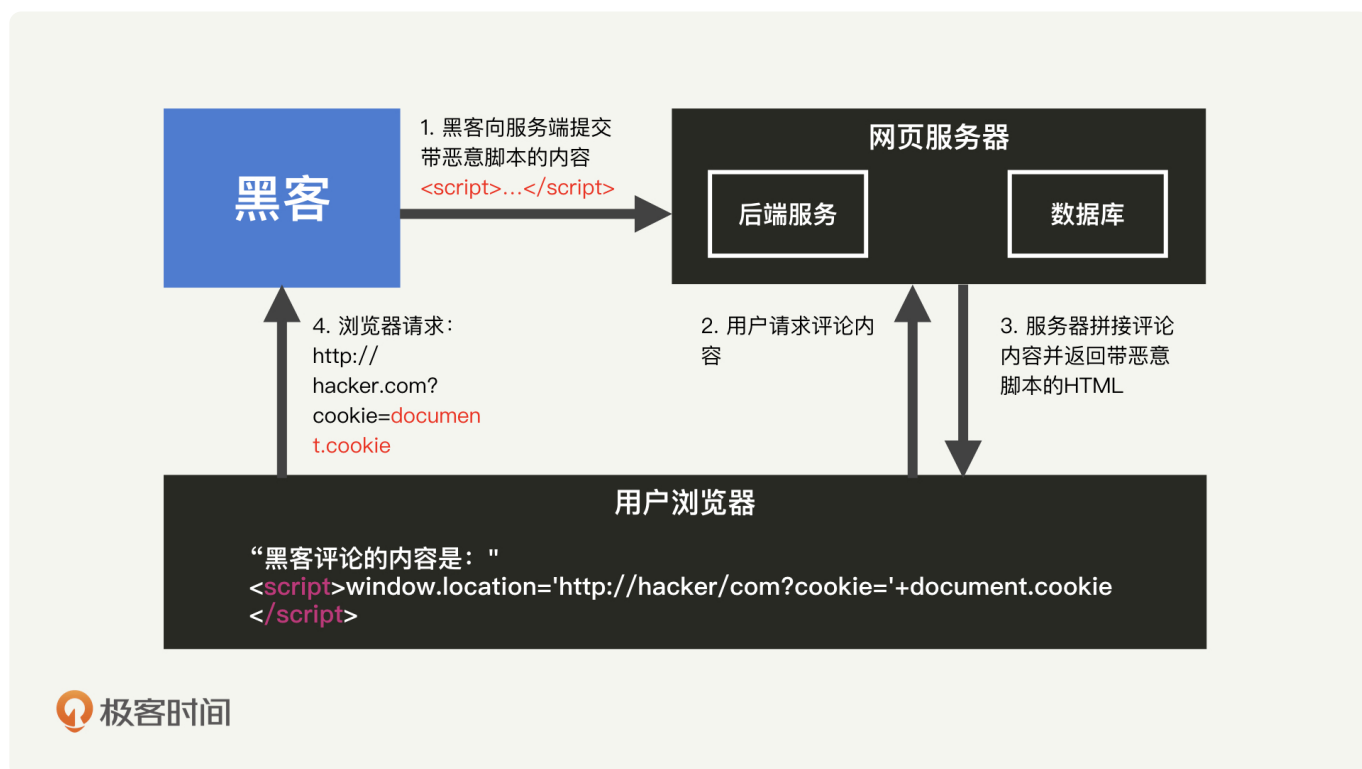
那么和上述例子一样，在基于 DOM 的 XSS 中，黑客也可以通过插入一段 `<script>alert('xss');</script>` 来执行指定的 JavaScript 脚本。基于 DOM 的 XSS 总体流程如下图所示。可以看到，这个流程其实和反射型 XSS 一致，只是不需要经过服务端了而已。



3. 持久型 XSS

你可以回想一下，当你在网页中搜索一个关键词时，实际上与这个关键词相关的所有搜索结果都会被展示出来。一旦这些搜索结果中，包含黑客提供的某个恶意 JavaScript 脚本，那么只要我们浏览了这个网页，就有可能会执行这些脚本。这就是持久型 XSS。因为这些恶意的搜索结果，会长期保存在服务端数据库中，所以它又叫作存储型 XSS。在应用中，存储用户的输入并对它们进行展示的地方，都可能出现持久型 XSS。比如：搜索结果、评论、博文等等。

有了前面的铺垫，持久型 XSS 的产生过程就很好理解了，具体我就不细说了，我还是把总体流程画了一张图，你可以仔细看看。



相比前面两种 XSS 攻击来说，持久型 XSS 往往具备更强的危害性。因为对于一个反射型或者基于 DOM 的 XSS 来说，需要黑客诱导用户点击恶意的 URL，才能够成功地在用户浏览器上执行 JavaScript 脚本。这对黑客在诱导用户操作方面的能力提出了考验：并不是所有的用户都是小白，一些有经验的用户会在点击链接前进行一定的考虑。

而持久型 XSS 则不同，它是将恶意的 JavaScript 脚本写入到了正常的服务端数据库中，因此，只要用户正常的使用业务功能，就会被注入 JavaScript 脚本。所以说，持久型 XSS 在传播速度和传播范围上，会远远超出其他类型的 XSS。

通过 XSS 攻击，黑客能做什么？

我们知道，这 3 种 XSS 攻击，都是因为黑客在用户的浏览器中执行了恶意的 JavaScript 脚本。那么执行这些 JavaScript 脚本有什么样的危害呢？我把这些危害总结了一下，可以分为下面几种。

1. 窃取 Cookie

从上面的例子中，我们可以看到，黑客可以窃取用户的 Cookie。因为黑客注入的 JavaScript 代码是运行在 `server.com` 这个域名下的，因此，黑客可以在 JavaScript 中通

过 `document.cookie` 获得 Cookie 信息。

另外，需要注意的是，受  **SOP** (Same Origin Policy, 同源策略) 保护，我们在 `server.com` 中是无法直接向 `hacker.com` 发送 GET 或者 POST 请求的。这也是为什么，在上面的例子中，我们需要通过 `window.location` 来执行跳转操作，间接地将 Cookie 信息发送出去。除了 `window.location` 之外，我们还可以通过加载 JavaScript 文件、图片等方式，向 `attacker.com` 发送带有 Cookie 的 GET 请求。

2. 未授权操作

除了窃取敏感信息以外，黑客还可以利用 JavaScript 的特性，直接代替用户在 HTML 进行各类操作。

在文章开头，我们提到的微博 XSS 攻击事件中，黑客就利用 JavaScript 脚本，让用户发送了一个微博，微博中同时还带有反射型 XSS 的链接。这样一来，每个点击链接的用户都会通过微博的形式，诱导更多的用户点击链接，一传十、十传百，造成大范围的传播。

3. 按键记录和钓鱼

窃取 Cookie 和未授权操作都是我们很容易想到的危害，除此之外，JavaScript 还能做什么呢？

JavaScript 的功能十分强大，它能够记录用户在浏览器中的大部分操作。比如：鼠标的轨迹、键盘输入的信息等。也就是说，你输入的账号名和密码，都可以被 JavaScript 记录下来，从而被黑客获取到。

另外，即使某个存在 XSS 漏洞的页面不具备任何输入框，黑客还可以通过修改 DOM，伪造一个登录框，来诱导用户在本不需要登录的页面，去输入自己的用户名和密码。这也是“钓鱼”的一种形式，在这个过程中用户访问的域名是完全正常的，只是页面被篡改了，所以具备更高的迷惑性。

如何进行 XSS 防护？

认识到 XSS 的危害之后，作为开发人员，我们最应该掌握的是，如何避免在开发过程中出现 XSS 漏洞。接下来我们就来看一看，具体有哪些防护方法。

1. 验证输入 OR 验证输出

防护的核心原则是：**一切用户输入皆不可信**。你的第一反应一定是，这很好实现啊，当接收到用户的输入时，我们就进行验证，这不就做到了吗？实际上并不是这么简单的，我们还是通过搜索这个例子来看。在用户点击“搜索”按钮之后，如果我们马上对他输入的内容进行验证，这样就会产生两个问题。

1. 你将无法保存用户的原始输入信息。这样一来，当出现了 Bug 或者想要对黑客行为进行溯源时，你只能“推断”，而不能准确地获取用户的原始输入。
2. 用户的内容可能会被多种语言获取和使用，提前编码或者处理，将产生未知的问题。比如，在旧版本的 PHP 中，就存在“[magic quotes](#)”的漏洞，因为 PHP 无法处理某些编码的字符而导致崩溃。

因此，我更推荐在需要输出的时候去进行验证，即当需要展示的时候，我们再对内容进行验证，这样我们就能够根据不同的环境去采取不同的保护方案了。

在 HTML 中，常见的 XSS 注入点我已经总结好了，你可以看下面这个表格：

注入点	示例
HTML元素内容	<code><div>内容用户</div></code>
HTML元素属性	<code><input value="用户内容"></code>
URL请求参数	<code>http://server.com?search=用户内容</code>
CSS值	<code>color: 用户内容</code>
JavaScript	<code>var name = "用户内容"</code>

2. 编码

现在，我们已经理解了，**XSS 防护的核心原则就是验证**，那具体该怎么去做验证呢？我认为，我们可以优先采用编码的方式来完成。所谓编码，就是将部分浏览器识别的关键词进行转换（比如 < 和 >），从而避免浏览器产生误解。对于客户端来说，编码意味着，使用 JavaScript 提供的功能对用户内容进行处理。具体的方法我也总结了一下，你可以看这个表格。

注入点	方法
HTML元素内容	node.textContent=用户输入
HTML原属属性	element.setAttribute('input', 用户输入) 或者 element['input']=用户输入
URL请求参数	window.encodeURIComponent(用户输入)
CSS值	element.style.color=用户输入



对于最后一个注入点，即在 JavaScript 中进行注入，目前还没有内置的编码方式来对它提供保护。你当然可以通过诸如 URL 编码等方式进行编码，但这有可能对应用的自身逻辑产生影响。因此，JavaScript 中的注入并不适合通过编码来进行保护。

3. 检测和过滤

但是，在很多时候，编码会对网页实际的展现效果产生影响。比如，原本用户可能想展示一个 1>0，却被编码展示成了 1>0。尽管网络环境安全了，却对用户造成了困扰。那么，我们还可以采取哪些方法进行验证呢？接下来我就为你介绍一下检测和过滤。

首先，我们需要对用户的内容进行检测。在这里，我们可以采用黑名单和白名单的规则。黑名单往往是我们最直接想到的方法：既然黑客要插入<javascript>标签，那么我们就检测用户内容中是否存在<javascript>标签就好了。

但是，黑客的攻击方法是无穷无尽的。你检测了<javascript>，黑客就可以改成<JavaScript>（因为 HTML 标签对大小写不敏感），甚至有些时候还能够编码成javascript等等。另外，HTML5 的发展速度很快，总是有新的标签被开发出来，这些新标签中也可能包含新的注入点。因此，黑名单的更新和维护过程，是需要我们和黑客进行长期对抗的过程

所以，在检测中，**我更推荐使用白名单的规则**。因为白名单的规则比较简单，并且十分有效。比如，在只输入一个分数的地方，规定只有整型变量是合法的。这样一来，你就能够检测出 99.99% 的攻击行为了。

说完了检测，那当发现某个用户的内容可能存在 XSS 攻击脚本时，我们该怎么处理呢？这个时候，处理选项有两个：拒绝或者过滤。毫无疑问，拒绝是最安全的选项。一旦你发现可能的 XSS 攻击脚本，只要不将这段用户内容展现出来，就能避免可能的攻击行为。

但是，拒绝会阻碍用户的使用流程，从用户体验的角度上来考虑的话，过滤会更被用户所接受。上面提到的编码就属于一种过滤的方式。除此之外，我们也可以直接对敏感字符进行替换删除等。需要注意的是，在替换的时候，一定不能采取黑名单的形式（比如：将 javascript 进行删除，那黑客就可以通过 JavaScript 来绕过），而是**应该采取白名单的形式**（比如，除了 div 之外的标签全部删除）。

同样地，**过滤的流程也必须彻底**。比如，我看到过有人采用下面这行字符串来过滤 javascript 标签：

```
1 $str=str_replace('<javascript>','', $str);
```

 复制代码

但黑客只需要将 str 的值变成<java<javascript>script>就可以了，因为 str_replace('<javascript>','', '<java<javascript>script>')的结果就是<javascript>。

4.CSP

面对 XSS 这样一个很普遍的问题，W3C 提出了 CSP（Content Security Policy，内容安全策略）来提升 Web 的安全性。所谓 CSP，就是在服务端返回的 HTTP header 里面添加

一个 Content-Security-Policy 选项，然后定义资源的白名单域名。浏览器就会识别这个字段，并限制对非白名单资源的访问。

配置样例如下所示：

 复制代码

```
1 Content-Security-Policy:default-src 'none'; script-src 'self';  
2 connect-src 'self'; img-src 'self'; style-src 'self';
```

那我们为什么要限制外域资源的访问呢？这是因为 XSS 通常会受到长度的限制，导致黑客无法提交一段完整的 JavaScript 代码。为了解决这个问题，黑客会采取引用一个外域 JavaScript 资源的方式来进行注入。除此之外，限制了外域资源的访问，也就限制了黑客通过资源请求的方式，绕过 SOP 发送 GET 请求。目前，CSP 还是受到了大部分浏览器支持的，只要用户使用的是最新的浏览器，基本都能够得到很好的保护。

总结

好了，我们讲了 XSS 的攻击类型、会产生的影响，以及如何对它进行防护。下面我来带你总结回顾一下，你要掌握的重点内容。

简单来说，XSS 就是利用 Web 漏洞，在用户的浏览器中执行黑客定义的 JavaScript 脚本，这样一种攻击方式。根据攻击方式的不同，可以分为：反射型 XSS、基于 DOM 的 XSS 和存储型 XSS。通过在用户的浏览器中注入脚本，黑客可以通过各种方式，采集到用户的敏感信息，包括：Cookie、按键记录、密码等。

预防 XSS 主要通过对用户内容的验证来完成。首先，我推荐在需要展示用户内容的时候去进行验证，而不是当用户输入的时候就去验证。在验证过程中，我们优先采用编码的方式来完成。如果编码影响到了业务的正常功能，我们就可以采用白名单的检测和过滤方式来进行验证。除此之外，我们可以根据业务需要，配置合适的 CSP 规则，这也能在很大程度上降低 XSS 产生的影响。

另外，在这里，我把本节课的重点内容梳理了一个脑图。你可以根据它来查漏补缺，加深印象。



思考题

好了，通过今天的学习，相信你已经了解了什么是 XSS 攻击。你可以试着分析一下，文章开头提到的事件中，黑客是利用哪种类型的 XSS 发起的攻击呢？我们应该怎么进行防御呢？

另外，在事件中我也描述了开发“甩锅”的场景：前端也好、后端也好，开发人员都不认为是自己的问题。那么，你认为出现这种安全事件，应该由谁来“背锅”呢？是开发、运维还是安全负责人呢？

欢迎留言和我分享你的思考和疑惑，也欢迎你把文章分享给你的朋友。我们下一讲再见！

点击查看 

来参加打卡，攻克 工作中 80% 的安全问题



PC端用户扫码参与



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 访问控制：如何选取一个合适的数据保护方案？

下一篇 07 | SQL注入：明明设置了强密码，为什么还会被别人登录？

精选留言 (10)

 写留言



return

2019-12-20

学到了，很透彻，写代码的时候得注意啊。

展开 



 3



冯玉鹏

2019-12-20

关于问题，到底应该谁背锅，应该是一个系统架构层面的问题，而不是单纯的前后端，如果一定要定位到前后端我认为应该是都有问题。

展开 



 1



qinsi

2019-12-22

不能只在输出时验证用户数据吧？比如攻击者提交了sql注入的数据，不去验证直接存数据库，然后就没有然后了

展开 ▾

作者回复: 这一讲只关注xss，sql注入是下一讲的内容哈～



我来也

2019-12-20

字符串替换的那个，如果不是替换为空字符串，而是替换成一个空格，那是不是就安全了呢？

作者回复: 也许是，不过最终的安全性，还是得看具体的场景。比如，万一替换了过后，还加了个strip呢。。



makawoyi

2019-12-20

反射型XSS，老师我这里的疑问 最开始是如何进行传播的呢？由黑客本人编写一条URL诱使用户点击，之后获取到用户的Cookie后通过JavaScript脚本再自动发送一条微博进行传播吗

安全事件应当加强人员安全意识，之后时尽可能使用上述提出的安全防护方式进行防护

展开 ▾

作者回复: 是的，就是通过发微博的形式进行再次传播。



tt

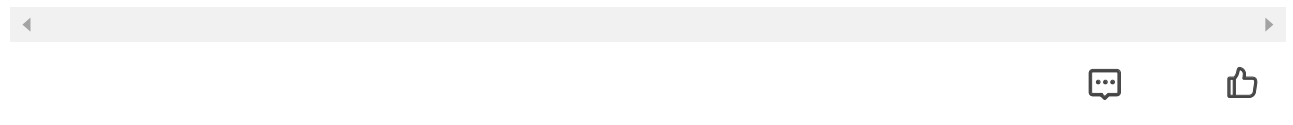
2019-12-20

不要相信用户的输入。那么前端要处理自用户的输入么？要。后端要处理用户的输入么？也要。没有正确处理，就有责任。

微博的例子是不是存储行XSS攻击呢？A用户发送的微博中被嵌入了恶意脚本，另外一个用

户B浏览此内容的时候，因为已经登陆了，可以直接用自己的身份（Token或cookies）直...
展开 ▾

作者回复: 不算存储型XSS，因为传播是因为微博的内容是一个反射型XSS链接。而这个内容本身实际不会执行脚本，需要用户点击才能发起XSS。



fuyuyu

2019-12-20

- 1.持久型 XSS；显示时验证输出，对用户的请求进行白名单检测和过滤，再加上csp。
- 2.安全维护人人有责，相关人员都应意识到安全的重要性，预防胜于治疗。



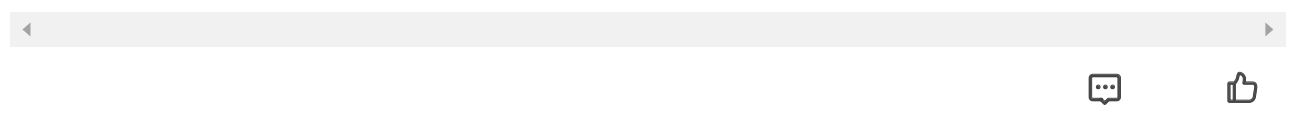
小晏子

2019-12-20

是这分析一下这次攻击，用户点击了链接就自动发送微博了，那么整个链路就是前端—后端—前端，而要达到这样的目的，一定是有javascript脚本被执行了，伪装的js脚本获取本地cookie后直接调用后端接口来发送微博，正好符合反射XSS的定义。对于这个问题，感觉开发运维安全都有责任，开发为什么没做安全字符检验，运维安全为什么没有对这种突然爆发的转发信息没有意识到安全问题

展开 ▾

作者回复: 责任肯定是都有的。问这个问题也是希望引导大家思考下安全在公司中的定位～



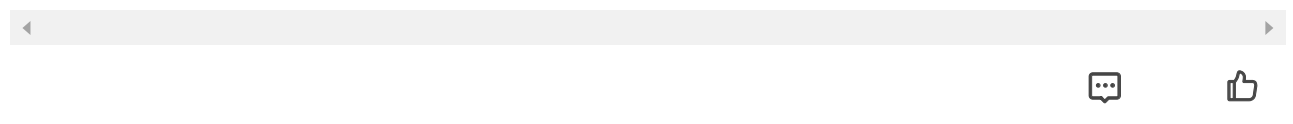
宝仔

2019-12-20

我觉得不应该去讨论背锅这个问题，问题核心在于怎么解决，以及各端明确未来的防范措施，下次再犯，再来定位锅的问题

展开 ▾

作者回复: 责任肯定是各方都有的。但锅也得有人背。尤其是如果事件影响大到某个程度，比如损失了10%的用户或者好几百万，肯定是需要有人受到惩罚的。。。



Cy23

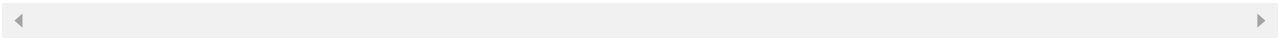


2019-12-20

我好像学到点怎么攻击别人，你们呢

展开 ∨

作者回复: 同学，未经授权发起攻击有法律风险，建议不要冒险～



💬 1

